

# ScoreFlow.

## Aplicación web para validar sistemas de reconocimiento de partituras musicales(OMR)

Marc Martín Martínez

**Resumen**– Este proyecto surge de la necesidad de corregir partituras musicales en formato MusicXML creadas a partir de *Optical Music Recognition* (OMR). El usuario sube a la aplicación un archivo MusicXML junto con una o varias imágenes de la partitura que usará como *ground truth* para revisar la pieza musical. A continuación, se muestra el grupo de imágenes que se hayan subido junto con un conjunto de compases establecidos por el archivo MusicXML. Se puede navegar libremente entre las diferentes imágenes que haya subido y los diferentes grupos de compases existentes. Usando una barra de herramientas, el usuario puede añadir, cambiar o eliminar: notas, accidentes, silencios... y cambiar características del compás: armadura, clave y signatura de tiempo. Finalmente, cuando se haya terminado de corregir la partitura, el usuario puede descargarse un nuevo archivo MusicXML con todos los cambios que haya hecho previamente.

**Palabras clave**– Partitura, Corrección, OMR, MusicXML, NodeJS, Vue.js, MySQL.

**Abstract**– This project comes up from the necessity of correcting musical scores in MusicXML format that were created from OMR *Optical Music Recognition*. The user uploads to the application a MusicXML file along with one or more images of the score that will be used as a ground truth to review the musical piece. Then, the group of images that have been uploaded are shown along with a set of measures that are established by the MusicXML file. You can freely navigate between the different images that were uploaded and the set of measures that exist. Using a toolbar, the user can add, modify or remove: notes, accidentals, rests... and change characteristics of the measure: key signature, clef and time signature. Finally, when the score is corrected, the user can download a new MusicXML file with all the changes that were made.

**Keywords**– Score, Correction, OMR, MusicXML, NodeJS, Vue.js, MySQL.

## 1 INTRODUCCIÓN

DURANTE la década de los 50 se funda la primera compañía dedicada al desarrollo y venta de dispositivos OCR (*Optical Character Recognition*), creada por David H. Shepard[1], desarrollador del primer dispositivo OCR. Con el paso de los años surge un nuevo

concepto de reconocimiento de caracteres, OMR (*Optical Music Recognition*), dedicado a la lectura de elementos musicales. La primera aplicación desarrollada es MIDISCAN, por *Musitek Corporation*[2]. Actualmente, hay varios investigadores dedicados a la mejora de la tecnología de reconocimiento y a la divulgación de esta, como se puede observar en la página *OMR Research*[23], y en los trabajos [19], [20], [21] y [22].

Aunque la tecnología de OMR ha avanzado, aún existen errores de lectura por parte del software de reconocimiento que da lugar a resultados incorrectos. Dada esta inexactitud, aparece la necesidad de poder editar estos archivos XML. Editar archivos XML directamente puede ser engo-

---

• E-mail de contacto: marcmartin315@gmail.com  
 • Mención realizada: Ingeniería del Software  
 • Trabajo tutorizado por: Dra. Alicia Fornés Bisquerra (Ciencias de la Computación)  
 • Curso 2019/20

roso, además de que los usuarios que usan la tecnología OMR no tienen por que tener conocimientos relacionados con XML. En cambio, es muy probable que los usuarios si tengan un entendimiento del funcionamiento de las partituras a nivel musical. Por lo tanto, este proyecto tiene el objetivo de aportar una herramienta auxiliar en la tecnología OMR que ayude a estos usuarios a editar los archivos XML utilizando sus competencias musicales sin necesitar la ayuda de terceros.

## 2 ESTADO DEL ARTE

Existen varias aplicaciones web con funcionalidades similares a la de este proyecto, aunque no dedicado exclusivamente a corregir archivos creados a partir de OMR, como *float.io*[3] y *Noteflight*[4].

Con respecto a *float.io*, su aplicación es capaz de representar muchos mas elementos musicales que la de este proyecto. Además da la opción de añadir varias voces a la partitura y asignarles instrumentos, para así poder reproducir la partitura desde la propia aplicación.

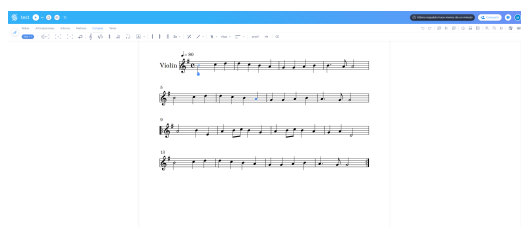


Fig. 1: Aplicación float.io

Aunque en esos aspectos es superior, no cuenta con un visualizador de imágenes. Que la aplicación de este proyecto si cuenta con ello proporciona al usuario una forma de corregir su partitura de una manera mucho mas cómoda y sencilla.

## 3 OBJETIVOS

En esta sección se explicarán los objetivos tenidos en cuenta para poder realizar este proyecto:

- Visualizar la partitura representando un conjunto de compases a partir del archivo MusicXML provisto por el usuario.
- Editar la partitura a partir de una interfaz de usuario que permita añadir, cambiar o eliminar elementos de la partitura.
- Creación de una base de datos donde se almacenen todos los usuarios de la aplicación y sus proyectos correspondientes.
- Desarrollar un entorno donde el usuario pueda crear nuevos proyectos y seleccionar los ya existentes para trabajar con ellos. Cada proyecto representa una partitura: imágenes de la partitura, archivo MusicXML y un nombre que identifique a dicho proyecto.
- Crear un nuevo archivo MusicXML con todos los cambios realizados por el usuario anteriormente. Estos

cambios también se guardarán en el servidor para el caso en el que el usuario quiera volver a modificar el mismo archivo en un futuro.

## 4 PLANIFICACIÓN

La planificación del proyecto se divide en tres fases. El diagrama de Gantt correspondiente se encuentra en la figura 2. Para una versión más detallada puede encontrarse la figura 11 en el apéndice. Ambos diagramas se crean con la aplicación de código abierto *GanttProject*. [8]

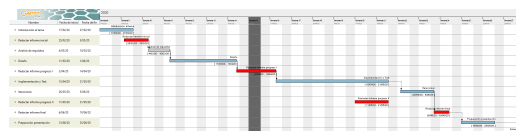


Fig. 2: Diagrama de Gantt.

A continuación se describe brevemente en que consistía cada fase:

- **Introducción al tema:** Esta fase consistía en hacer un primer contacto con la materia. Se estudió el funcionamiento de los archivos MusicXML, se comprobó cual era la situación actual referente al OCR y OMR, además de aplicaciones similares al proyecto, y se creó un glosario de conceptos musicales básicos. Este último fue útil en las primeras fases del proyecto donde aún no se dominaba completamente los conocimientos musicales requeridos para crear partituras coherentes.
- **Análisis y diseño:** Identificación y clasificación de requerimientos funcionales y no funcionales. Diseño del software mediante diagramas UML y diseño de la interfaz de usuario con prototipo y persona.
- **Implementación:** Escritura del código respecto a los requerimientos anteriores y testing. Definición de iteraciones en la fase final del proyecto para hacer *user testing*.

## 5 METODOLOGÍA

El proyecto ha seguido un modelo **iterativo** desde su inicio hasta su fin. A medida que el proyecto avanzaba en el tiempo se iban modificando elementos del mismo, como por ejemplo el prototipo anteriormente mencionado. El diseño del programa también iba sufriendo pequeños cambios, al ver que surgían nuevas necesidades en la interfaz de usuario o se detectaban características obsoletas o daban lugar a dudas.

## 6 CONCEPTOS BÁSICOS




En esta sección se hará un breve repaso al funcionamiento y peculiaridades de las partituras musicales y de los archivos MusicXML. **Solo se harán mención** a los elementos que tiene en cuenta la aplicación para su funcionamiento.

## 6.1. Partitura Musical

La partitura podemos dividirla en dos conceptos principales: compases y los atributos de estos.

### 6.1.1. Atributos

Definen como se construyen los compases y los modifican. Siempre se encuentran al principio del pentagrama.

- **Armadura:** Se representa mediante los caracteres  $\flat$  (bemol) o  $\sharp$  (sostenido). Dependiendo el número de estos elementos el pentagrama sigue una tonalidad u otra. Este número debe de estar dentro del rango [0,7]. No pueden haber combinaciones de sostenido y bemol.
- **Clave:** Signo que determina la altura, más o menos aguda/grave, de la música representada. Las principales son la clave de Sol ()<sup>1</sup>, Fa () y Do ()<sup>2</sup>. Dependiendo en que línea del pentagrama se dibujen estos signos la clave será de un tipo u otro.
- **Signatura de Compás:** Fracción que determina cuantos pulsos debe de tener un compás. El denominador representa la duración de cada pulso y el numerador el número de pulsos que ha de haber en el compás. Los más comunes son :  $\frac{2}{4}$ ,  $\frac{3}{4}$ ,  $\frac{4}{4}$ ,  $\frac{2}{8}$ ,  $\frac{6}{8}$ .

### 6.1.2. Compases

Contienen todo tipo de elementos musicales: notas, silencios, barras, etc. Las notas y silencios pueden durar desde 4 pulsaciones (redonda) hasta  $\frac{1}{64}$  de pulsación (semifusa).

- **Notas:** Representación que puede tomar varias formas para determinar cuantos pulsos durará el sonido que representa dentro del compás. Dependiendo en que línea del pentagrama se dibuje, el sonido será más grave o más agudo, siendo las líneas más bajas las más graves y las más altas las más agudas. Las notas pueden sufrir todo tipo de modificaciones mediante otros elementos musicales: **accidentes** y **articulaciones**.
- **Silencios:** Símbolo que indica al músico que no debe de reproducir ningún sonido durante la duración que represente la forma del silencio.
- **Accidentes:** Al igual que en la armadura, se representan con  $\flat$ ,  $\sharp$  y adicionalmente  $\natural$  (becuadro). Los sostenidos incrementan en un semitono<sup>1</sup> la altura de la nota afectada, mientras que el bemol lo disminuye. El becuadro cancela cualquier efecto sobre la nota de un sostenido o un bemol.
- **Articulaciones:** Modifican la manera de interpretar las notas:
  - **Ligadura:** Línea que une dos notas a la misma altura, indicando que la primera nota ha de durar la suma de la duración de las dos notas unidas. La segunda nota no se interpreta posteriormente.
  - **Legato:** Línea que agrupa dos o más notas en diferentes alturas, indicando al músico que ha de reproducir el sonido de ellas sin interrupción.

- **Puntillo:** Punto que se añade a una nota para alargar su duración un medio más de lo que duraba anteriormente ( $\times 1.5$ ).
- **Calderón:** Símbolo añadido a una nota que indica al músico que la duración de la nota se ha de alargar. Este alargamiento depende de la interpretación del músico, a diferencia del puntillo que es una medida exacta.

<sup>1</sup>Todas las notas tienen un tono de distancia entre ellas, a excepción de Mi-Fa y Si-Do que tienen un semitono.

## 6.2. MusicXML

Archivo compuesto enteramente por *tags*. En esta sección se mencionarán los tags más relevantes para el funcionamiento de la aplicación. Inicialmente el documento ha de determinar cuantas voces tiene la partitura. Una voz dentro de una partitura representa un instrumento. Una partitura puede tener una o más voces. La aplicación solo puede interpretar una única voz.

Mediante un identificador, se crea el tag `<part id=idVoz>` para indicar que todo lo que contenga el tag hace referencia a esa voz. Cada compás se representa con el tag `<measure>`. Este contiene los tags `<note>` y `<attributes>` siendo los más importantes. El tag **attributes** contiene los tags:

- **Key:** Contiene los tags `<fifths>` y `<mode>` para representar la armadura del compás. Fifths puede tomar los valores [-7, 7], indicando el número de  $\flat$  y  $\sharp$ . Los números negativos corresponden a bemoles y los positivos a sostenidos. Mode indica si la escala es mayor o menor, entre otros que están en desuso.
- **Clef:** Contiene los tags `<sign>` y `<line>` para representar la clave. Sign determina si la clave será de Sol, Fa o Do. Line especifica en que línea debe dibujarse la clave para indicar el tipo de clave.
- **Time:** Contiene los tags `<beats>` y `<beat-type>` para representar la signatura de compás. Beats representa el numerador y Beat-type el denominador.
- **Divisions:** Determina el número por el cual se ha de dividir el tag de duration de una nota para poder obtener su duración en pulsos.

Asimismo, el tag **note** contiene los tags:

- **Pitch:** En él se encuentran los tags `<step>` y `<octave>`. Define en que línea del compás ha de dibujarse la nota.
- **Duration:** Número que se utiliza para calcular la duración en pulsos de la nota.
- **Type:** String que determina la duración de la nota.
- **Rest:** Si la nota contiene este tag pasa a ser un silencio.
- **Accidental:** Añade un accidente a la nota dependiendo del string que contenga el tag.

- **Articulations:** Contiene el tag de fermata, que representa al calderón.
- **Dot:** Si la nota contiene este tag esta tendrá un puntillo.
- **Notations:** Contiene el tag slur para representar ligaduras y legatos. Slur tiene el atributo type que determina si es el principio de la ligadura o el final.

## 7 ANÁLISIS DE REQUISITOS

La meta de esta fase era identificar las funciones básicas que necesitaba el proyecto y como implementar estas de la manera más óptima posible. Listar y clasificar los requisitos en funcionales y no funcionales, como puede verse en las tablas 1 y 2.

Funcionales	
ID	Descripción
RF-1	Subir un archivo MusicXML a la aplicación y su correspondiente imagen de la partitura escaneada.
RF-2	Visualizar la partitura e imagen en la pantalla.
RF-3	Seleccionar elemento que se quiere modificar con un click de ratón.
RF-4	Seleccionar notación musical: anglosajona (A, B, C, D, E, F, G) o latina (do, re, mi, fa, sol, la, si).
RF-5	Añadir, modificar o eliminar nota al compás, seleccionando <i>tempo</i> y nota.
RF-6	Añadir o quitar a nota existente un puntillo o calderón.
RF-7	Añadir una ligadura entre dos notas.
RF-8	Añadir o quitar accidente a una nota: sostenido (#), bemol (b) y becuadro (b).
RF-9	Seleccionar tipo de armadura. Puede tenerla o no.
RF-10	Seleccionar clave: Sol (♩), Fa (♭) y Do (♮).
RF-11	Seleccionar la signatura de compás: $\frac{4}{4}$ , $\frac{3}{4}$ , etc.

ID	Descripción
RF-12	Añadir, modificar o eliminar silencios.
RF-13	Cambiar tipo de barra del compás: simple, doble o de repetición.
RF-14	Exportar cambios realizados en la partitura a un archivo MusicXML.

TABLA 1: REQUERIMIENTOS FUNCIONALES

No funcionales	
ID	Descripción
RNF-1	El sistema debe de usar mensajes de error que describan el problema de manera precisa y que el usuario final pueda entenderlo (sin tecnicismos).
RNF-2	El feedback de la interfaz de usuario debe producirse en menos de 1 segundo. En caso de que sea mayor, indicar al usuario de que su petición se está cargando.
RNF-3	La aplicación debe de contar con un diseño <i>responsive</i> .
RNF-4	Uso de diálogos de confirmación para evitar acciones accidentales por parte del usuario en los que es difícil recuperar el estado anterior a la acción.

TABLA 2: REQUISITOS NO FUNCIONALES

Después, usando el modelo de Kano, se volvieron a clasificar en *dissatisfiers*, *satisfiers* y *delighters*. Las tablas correspondientes similares a las anteriores pueden encontrarse en las tablas 5, 4 y 3 del apéndice.

## 8 DISEÑO

Creación de elementos considerados útiles para poder hacer un buen diseño de la aplicación.

- **Diagramas UML**

Creación de diagramas de casos de uso, de secuencia y de clase, como por ejemplo las figuras 3 y 4. En el apéndice se pueden ver imágenes ampliadas de estas figuras para verse mejor. Otros diagramas pueden verse en la sección del apéndice: Diseño. Utilización de *VisualParadigm* [9] para componer estos diagramas.



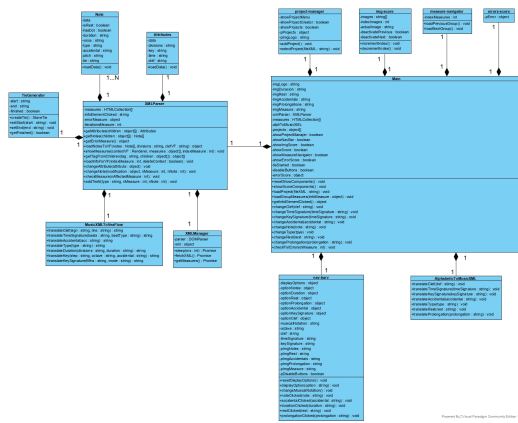


Fig. 3: Diagrama de Clase

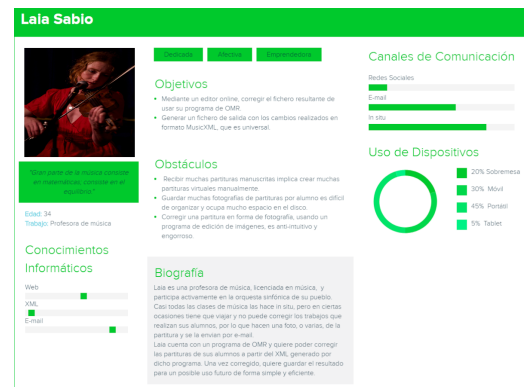


Fig. 5: Persona

## 9 IMPLEMENTACIÓN

La planificación inicial de esta fase consistía en visualizar la partitura y después poder editarla, para finalmente exportar el nuevo archivo MusicXML. Todo esto siguiendo el paradigma de testing *Test-Driven Development*. Una vez acabado esto se esperaba tener hasta 3 iteraciones de una semana de duración cada una. En estas iteraciones se detectarían errores mediante *user testing* con usuarios con conocimientos musicales para después modificar requisitos y/o el diseño, e implementar estos cambios. Esta fase sufrió algunos cambios con respecto lo planificado al principio del proyecto por una planificación demasiado optimista, no teniendo suficiente tiempo para las iteraciones.

## 10 RESULTADOS

La aplicación está desarrollada en Nodejs. Esta está dividida en *backend*, la parte del servidor, y *frontend*, la del cliente.

### 10.1. Backend

En este apartado se mencionan las funcionalidades desarrolladas en el servidor y su funcionamiento. Se utilizan las extensiones:

- *Express*[12]. Framework para crear el servidor y hacer *routing*.
- *Express-fileupload*[13]. Módulo para *express* para la subida de archivos del cliente al servidor.
- *Express-session*[14]. Módulo para la gestión de sesiones que identifican a los usuarios.
- *bcrypt*[15]. Módulo para encriptar cadenas de caracteres. Utilizado para cifrar la contraseña de los nuevos usuarios y para comprobar que la contraseña de un usuario que se *loggea* es la misma que en la base de datos.
- *mysql*[16]. Driver para conectarse a una base de datos MySQL y hacer consultas en ella.

El servidor tiene dos propósitos principales:

1. Servir los archivos o redirigir al usuario según las peticiones que haga el cliente. Posible gracias al framework mencionado anteriormente, *express*.

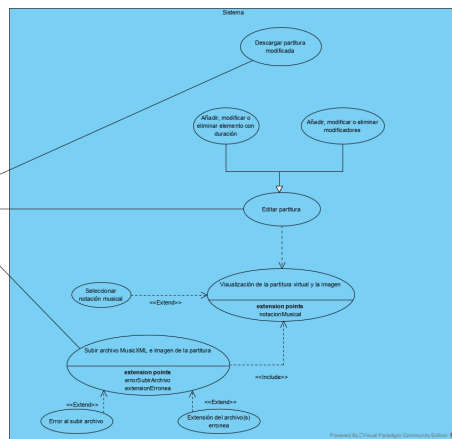


Fig. 4: Diagrama de Casos de Uso

- **Prototipo**  
Creación de un prototipo simple para representar el editor de partituras usando la herramienta de edición de imágenes digitales *GIMP*[10]. Este prototipo sería desechado más adelante, ya que se decidiría usar una barra de herramientas para la edición de la partitura. El prototipo está disponible en el apéndice, en la página 10.
- **Persona**  
Desarrollo de una *persona* para facilitar que la interfaz del usuario estuviera enfocada en un posible usuario de la aplicación, y no en el desarrollador de la aplicación. Utilización de la herramienta *Xtensio*[11] para crear la persona, representada en la figura 5.

2. Comunicarse con la base de datos para incluir nueva información o solicitarla.

La clase **DataBase** alberga toda la información necesaria para acceder a la base de datos: usuario, contraseña, host y nombre de la base de datos. La clase cuenta con varias funciones para hacer consultas sql contra la base de datos, como crear usuarios o proyectos u obtenerlos. El diagrama entidad-relación puede verse en la figura 6.

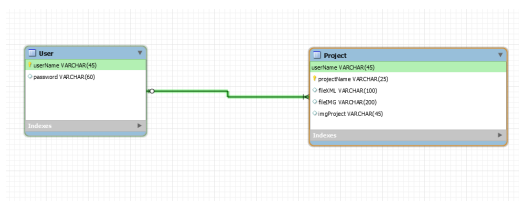


Fig. 6: Diagrama Entidad-Relación de la base de datos.

Para hacer el diagrama anterior y diseñar y crear la base de datos se utilizó el programa *MySQL Workbench*[17].

## 10.2. Frontend

Esta sección muestra los elementos creados para el frontend. Para crear una interfaz de usuario reactiva a los cambios hechos por el usuario se utiliza el framework *Vue*.

### 10.2.1. SignIn y LogIn

Esta página contiene un formulario para que el usuario pueda crear una cuenta en la aplicación. A la hora de confirmar la contraseña de la nueva cuenta, la aplicación comprueba que las dos contraseñas añadidas sean iguales antes de enviarlas al servidor. En la figura 7 se puede observar el formulario para crear una nueva cuenta.

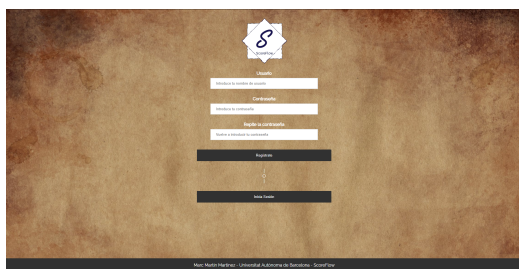


Fig. 7: Formulario de creación de usuario.

Asimismo, la página también incluye otro formulario para poder iniciar sesión. El usuario puede cerrar la sesión en cualquier momento una vez *loggeado* usando el botón en la esquina superior derecha.

### 10.2.2. Gestor de Proyectos

Este componente es el responsable de mostrar al usuario sus proyectos y de darle la opción de crear nuevos. Para poder crear un nuevo proyecto el usuario ha de seleccionar un nombre para el proyecto, un archivo XML, y la imagen o imágenes correspondientes a la partitura. El nombre del proyecto ha de ser único. Los proyectos se muestran usando el nombre del proyecto,

el nombre del archivo XML y una imagen de la partitura, como se puede ver en la figura 8.



Fig. 8: Gestor de Proyectos

Una vez el usuario selecciona un proyecto el gestor de proyectos desaparece y se muestra el editor de partituras. El usuario siempre puede volver al gestor de proyectos usando el logo en la esquina superior izquierda.

### 10.2.3. Visualización de la partitura

Se crea la clase **XMLParser**, responsable de representar la partitura a partir del archivo MusicXML y la de aplicar los cambios hechos por el usuario.

Inicialmente se lanza una petición al servidor para obtener el archivo MusicXML del usuario utilizando un *fetch*. Después, se parsea todo el documento para extraer los tags necesarios para crear una partitura. Dado que es un XML, se utiliza la interfaz *Document* [18] para poder modificar el árbol *DOM*. La información obtenida es almacenada en otras dos clases:

- **Attributes.** Contiene la información de los tags del compás: clave, signatura de tiempo y armadura. Esta clase solo puede tener una instancia, por lo que en una partitura solo puede haber una armadura, clave y signatura de tiempo. Es decir, el usuario no puede añadir nuevos atributos en mitad de la partitura. La aplicación solo tiene en cuenta el primer tag de atributos que ha de encontrarse en el primer compás.
- **Note.** Contiene la información de los tags de una nota: duración, tono y si es o no un silencio. Además de los elementos que se le pueden añadir a una nota: accidentes, *legatos* y ligaduras y prolongaciones (puntillo o calderón).

Para gestionar el funcionamiento de los *legatos* y ligaduras, que son representados de la misma forma dentro de la aplicación, se crea la clase **TieGenerator**. Su función básica es almacenar la primera y la última nota que forma una ligadura. Una vez se tienen las dos notas, crea un objeto de *VexFlow* que representa dicha ligadura. Además, controla si la clase *XMLParser* ha de mostrar un compás adicional en el caso de que la nota inicial y la final se encuentren en dos compases distintos.

Actualmente la aplicación solo puede mostrar correctamente una ligadura si el número de compases entre los que se encuentra el inicio y final de una ligadura es de uno. Además, una nota no puede pertenecer a una ligadura si esta está entre otras dos notas que forman una ligadura diferente. Por lo tanto, no pueden haber ligaduras dentro de otras

ligaduras.

Para poder representar toda esta información en forma de partitura se utiliza la API VexFlow [6]. Para poder construir la partitura, esta información ha de ser *traducida* para que la API pueda utilizarla. Para ello se crea otra clase, **MusicXMLToVexFlow**. En ella, se compara cada tag con todas las posibles opciones que puede tener el tag, que estén disponibles en la API, y devuelve ese equivalente.

Finalmente, se inicializan los valores necesarios para VexFlow: tamaño de los compases, del SVG(Scalable Vector Graphics) donde se encontrará la partitura... y se crean los objetos de VexFlow para después dibujarlos y así conseguir representar la partitura. La aplicación representa hasta 5 compases a la vez, 6 en el caso de una ligadura con notas en diferentes compases.

A cada objeto que representa una nota se le asignan los atributos *measure* y *note*, que representan el número compás donde se encuentra la nota y el índice de la nota dentro de ese compás. Además de *event listeners* a *clicks*. Esta información será usada más adelante cuando el usuario quiera modificar una nota.

Existen algunos elementos que la aplicación no es capaz de representar por que son poco utilizados.

#### Claves:

- Bass Clef (Down Octave)
- Vocal Tenor Clef

#### Armaduras fuera de la escala diatónica:

- Dorian                      ■ Mixolydian                      ■ Locrian
- Phrygian                      ■ Aeolian
- Lydian                      ■ Ionian

#### Armaduras teóricas:

- F $\flat$                                       ■ G $\sharp$
- d $\flat$                                       ■ e $\sharp$

#### Duración:

- Máxima                      ■ Longa                      ■ Cuadrada
- Garrapatea                      ■ Semigarrapatea

#### 10.2.4. Editor de Partituras

La interfaz se compone de: barra de herramientas, imagen o imágenes de la partitura, errores recogidos de la partitura y el navegador de partituras, que consiste en dos botones para avanzar o retroceder en el grupo de compases. Cada elemento de la interfaz de usuario se representa en componentes de *Vue*[7]. Estos componentes y la representación de la partitura pueden verse en la figura 9.

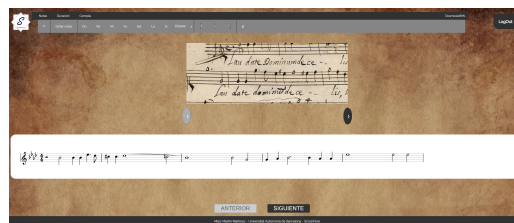


Fig. 9: Editor de Partituras

El componente *nav-bar*, o la barra de herramientas, permite al usuario modificar la nota *clickada* anteriormente o los atributos del compás. La tonalidad de una nota puede cambiarse utilizando uno de los botones que designa un tono más la octava que se quiera utilizar. Los accidentes y prolongaciones pueden añadir o quitarse. La duración puede modificarse desde la redonda, 4 pulsos, hasta la semifusa,  $\frac{1}{64}$  pulsos.

El usuario puede cambiar el modo en el que actúa el editor de partituras haciendo click en el botón de cambiar modo. Hay dos modos: editar las notas ya existentes, como se comentaba anteriormente, o añadir de nuevas. Una vez se ha seleccionado el modo de añadir notas, el usuario debe hacer click en una nota cualquiera del compás y seleccionar la tonalidad y octava de la nota que quiere añadir. La nueva nota aparecerá a continuación de la nota *clickada* anteriormente. Por defecto la nota creada tiene una duración de un pulso (negra).

Si en cambio desea eliminar una nota, solo ha de hacer click en ella y usar el botón de eliminar notas.

Con respecto al compás, puede modificarse: la clave, la signatura del compás y la armadura.

Adicionalmente, la barra de herramientas cuenta con dos modos de visualización de sus opciones: notación musical latina y anglosajona, para que el usuario pueda usar la notación con la que se sienta más cómodo. La aplicación funciona internamente con la notación anglosajona.

Para poder aplicar los cambios que haga el usuario al archivo MusicXML se ha de volver a traducir la información transmitida por el usuario al formato MusicXML. Para ello se crea otra clase, **AlphabeticToMusicXML**, muy similar a la anteriormente mencionada MusicXMLToVexFlow.

Para visualizar las imágenes subidas por el usuario se cuenta con otro componente que cuenta con dos botones para avanzar o retroceder en las diferentes imágenes, y la imagen que quiera visualizar el usuario. La imagen seleccionada puede ser ampliada para ayudar al usuario a leerla mejor. Para ello se puede hacer *zoom in* y *zoom out* utilizando la rueda del ratón y arrastras la imagen con el ratón. Se ha utilizado el módulo *vue-zoomer* [24] para poder ampliar la imagen.

En el caso que el usuario modifique un compás, y este no respete el número de pulsos necesario dictaminado por la signatura de compás, se mostrará un mensaje de error mostrando el número de pulsos actual del compás y el número necesario para que este sea correcto. Un ejemplo de este error se muestra en la figura 10.



Fig. 10: Mensaje de error del compás

Finalmente se muestra el conjunto de compases más otros dos botones para poder avanzar o retroceder a lo largo de la partitura.

## 11 TEST

En esta sección se describirá como se ha realizado el testing de la aplicación. Inicialmente, como estaba en la planificación, se seguiría el paradigma *Test-Driven Development*. Debido a retrasos en distintos puntos del desarrollo del proyecto, esta idea quedo omitida para poder dar prioridad a otros aspectos. Remplazando esta idea, se decidió que a medida que se añadiesen nuevas funcionalidades a la aplicación se concretarían una serie de escenarios posibles con la nueva funcionalidad. Después se probarían uno a uno para identificar posibles errores o confirmar el correcto comportamiento de dicha funcionalidad.

### 11.1. Test con usuario final

En el transcurso del proyecto se contó con la ayuda de un músico que utilizó la aplicación y dio su opinión para mejorar algunos aspectos de esta:

1. La visualización de las ligaduras/legatos no es del todo correcta. La línea que la forma no debería atravesar las notas que están en el camino de la nota inicial a la final.
2. Sería útil que al seleccionar una nota con un click se resaltase de alguna manera para saber que nota se está modificando.
3. Cuando hay un error en un compás sería útil que dicho compás quedase marcado de alguna manera para saber que se ha de corregir.
4. Los calderones típicamente se dibujan en la parte superior del compás, en la aplicación están en la parte inferior.
5. Hacer click en la imagen para hacer zoom hace que la imagen cubra la barra de herramientas y no se pueda modificar la partitura. Es engorroso tener que hacer zoom y después volver a quitarlo para seguir trabajando.

Respecto al punto 5, gracias a este feedback se rediseñó la forma de visualizar las imágenes por la actual, que facilita mucho el trabajo con la partitura con respecto a la anterior.

## 12 CONCLUSIONES

Una vez finalizado el proyecto, es correcto decir que se han alcanzado todos los objetivos propuestos. La aplicación es capaz de registrar nuevos usuarios y de dar acceso a aquellos que ya lo han hecho. Estos usuarios son libres de crear proyectos nuevos a partir de sus archivos MusicXML e imágenes y de trabajar con dichos proyectos. Los archivos MusicXML se representan correctamente en forma de partitura en la aplicación y el usuario es libre de modificar cualquier elemento de la partitura, incluido añadir o quitar elementos. Una vez el usuario cree conveniente que ha finalizado con el proyecto o quiere continuar en otro momento puede guardar los cambios. Una vez guardados se

exporta un nuevo archivo XML con los cambios realizados. Aunque es cierto que se han cumplido los objetivos, la aplicación tiene un gran margen de mejora:

- Añadir más elementos musicales que puedan ser representados en la partitura y editarlos o crear de nuevos.
- Poder escribir anotaciones en la propia partitura.
- Poder modificar los atributos del compás (armadura, clave y signatura de compás) en cualquier momento de la partitura, y no solo la inicial.
- Representar más de una voz en la partitura.
- Hacer que la aplicación permita editar una misma partitura de forma colaborativa con otros usuarios.

Y otras muchas más características, dado que la representación musical es muy extensa y variada.

Por otra parte, uno de los puntos débiles del proyecto es que la planificación, y el seguimiento de ella, no se ha hecho de la mejor manera posible. Esto ha implicado que no se haya dedicado el tiempo suficiente al testeo de la aplicación, como se mencionaba anteriormente en la sección de planificación. Otro factor que también ha afectado al desarrollo del proyecto de forma negativa ha sido subestimar la importancia de un diseño exhaustivo. Al no dedicar más tiempo a él, se crearon nuevas funciones o clases sobre la marcha, dando como resultado a una situación caótica en el código, desarrollando funciones poco eficientes o que al poco tiempo quedaban obsoletas, teniendo que reprogramar funcionalidades que se consideraban ya completas.

## AGRADECIMIENTOS

Me gustaría empezar esta sección dándole las gracias a mi padre y mi abuela, que sin ninguna duda no estaría siquiera haciendo este trabajo si no fuese por su apoyo durante todos estos años.

Darle las gracias también a Alicia, mi tutora, que ha estado siempre disponible para cualquier duda que tuviese incluso dada la situación tan peculiar de este último semestre. Además de que sus apuntes y consejos siempre han sido muy útiles y soy consciente que sin ellos el resultado de este trabajo no sería el mismo sin ella.

Gracias también a COC, ellos ya saben quienes son, que me han apoyado y dado ánimos desde el principio hasta el final de este proyecto.

Para acabar aprovecho también para agradecerles a Arnau Baró y Carles Badal por haberme provisto de archivos XML e imágenes de partituras que me han ayudado en gran medida a probar esta aplicación con su aportación.

## REFERENCIAS

- [1] David H. Shepard. Character reading techniques. <https://patents.justia.com/patent/4021777>
- [2] Musitek Corporation. SmartScore. <https://www.musitek.com/>
- [3] flat.io <https://flat.io>

- [4] *Noteflight*. <https://www.noteflight.com/>
- [5] *Make Music*. MusicXML. <https://www.musicxml.com/>
- [6] *Mohit Muthanna Cheppudira*. VexFlow [www.vexflow.com](http://www.vexflow.com)
- [7] *Evan You*. Vue.js <https://vuejs.org/>
- [8] *Alexandre Thomas and Dmitry Barashev* GanttProject <https://www.ganttproject.biz>
- [9] *Visual Paradigm*. <https://www.visual-paradigm.com/>
- [10] *GIMP*. <http://www.gimp.org/>
- [11] *Xtensio*. <https://xtensio.com/>
- [12] *OpenJS Foundation*. Express. [expressjs.com](https://expressjs.com)
- [13] *Richard Girges and Roman Burunkov*. Express-fileupload. <https://github.com/richardgirges/express-fileupload>
- [14] *Douglas Christopher Wilson*. Express-session. <https://github.com/expressjs/session>
- [15] *Nicholas Campbell*. Bcrypt. <https://github.com/kelektiv/node.bcrypt.js>
- [16] *Felix Geisendörfer and contributors* MySQL. <https://github.com/mysqljs/mysql>
- [17] *MySQL MySQL Workbench*. <https://www.mysql.com/products/workbench/>
- [18] Document. <https://developer.mozilla.org/en-US/docs/Web/API/Document>
- [19] *Arnau Baro-mas*. Recognition of handwritten music scores. [http://www.cvc.uab.es/people/afornes/students/TFG\\_2016\\_ABaro.pdf](http://www.cvc.uab.es/people/afornes/students/TFG_2016_ABaro.pdf)
- [20] *Arnau Baro-mas*. Optical Music Recognition by Long Short-Term Memory Recurrent Neural Networks. [http://www.cvc.uab.es/people/afornes/students/master\\_2017\\_ABaro.pdf](http://www.cvc.uab.es/people/afornes/students/master_2017_ABaro.pdf)
- [21] *Alicia Fornés Bisquerra*. Analysis of Old Handwritten Musical Scores. [http://www.cvc.uab.es/people/afornes/publi/AFornes\\_master.pdf](http://www.cvc.uab.es/people/afornes/publi/AFornes_master.pdf)
- [22] *Alicia Fornés Bisquerra*. Writer Identification by a Combination of Graphical Features in the Framework of Old Handwritten Music Scores. <http://www.cvc.uab.es/people/afornes/publi/PhDAliciaFornes.pdf>
- [23] *OMR Research*. <https://omr-research.net/>
- [24] *jarvisniu*. vue-zoomer. <https://github.com/jarvisniu/vue-zoomer>

## APÉNDICE

### A.1. Planificación

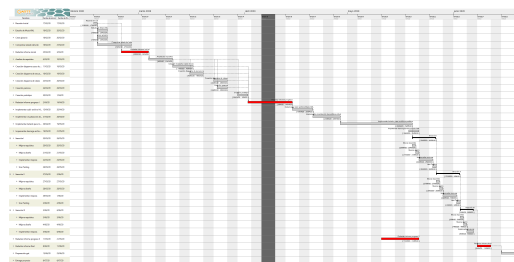


Fig. 11: Diagrama de Gantt detallado.

## A.2. Diseño

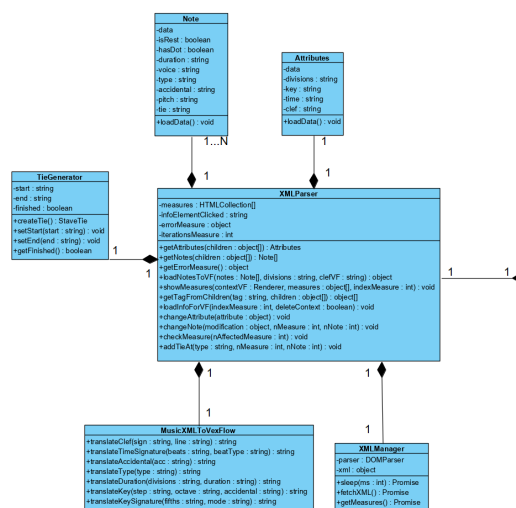


Fig. 12: Diagrama de Clases: Ampliado 1.

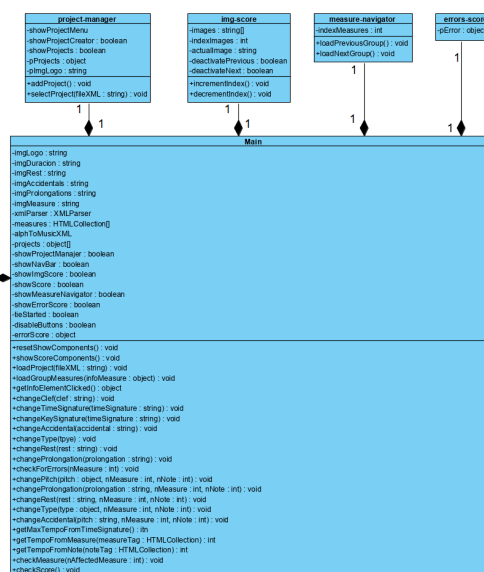


Fig. 13: Diagrama de Clases: Ampliado 2.





### A.3. Análisis de Requisitos




Dissatisfiers	
ID	Descripción
DS-1	Subir un archivo MusicXML a la aplicación y su correspondiente imagen de la partitura escaneada.
DS-2	Visualizar la partitura e imagen en la pantalla.
DS-3	Seleccionar elemento que se quiere modificar con un click de ratón.
DS-4	Añadir, modificar o eliminar nota al compás, seleccionando <i>tempo</i> y nota.
DS-5	Seleccionar tipo de armadura. Puede tenerla o no.
DS-6	Seleccionar clave: Sol (  ) Fa(  ) y Do(  ) y sus variantes.
DS-7	Seleccionar la signatura de compás: $\frac{4}{4}$ , $\frac{3}{4}$ , etc.
DS-8	Añadir, modificar o eliminar silencios.
DS-9	Exportar cambios realizados en la partitura a un archivo MusicXML.

TABLA 3: DISSATISFIERS SEGÚN EL MODELO DE KANO.

Delighters	
ID	Descripción
D-1	Seleccionar notación musical: anglosajona (A, B, C, D, E, F, G) o latina ( <i>do, re, mi, fa, sol, la, si</i> ).

TABLA 4: DELIGHTERS SEGÚN EL MODELO DE KANO.

Satisfiers	
ID	Descripción
S-1	Cambiar tipo de barra del compás: simple, doble o de repetición.
S-2	Añadir o quitar a nota existente un puntillo (·).
S-3	Añadir ligadura entre dos o más notas.
S-4	Añadir o quitar accidente a una nota: sostenido (#), bemol (♭) y becuadro (♮).

TABLA 5: SATISFIERS SEGÚN EL MODELO DE KANO.